User-specific Network Service Functions in an SDN-enabled Network Node

User-specific Network Service Functions in an SDN-enabled Network Node

Ivano Cerrato, Alex Palesandro, Fulvio Risso
Department of Computer and Control Engineering
Politecnico di Torino
Torino, Italy
Email: {ivano.cerrato, fulvio.risso}@polito.it
alex.palesandro@studenti.polito.it

Tobias Jungel, Marc Suñé, Hagen Woesner
Berlin Institute for Software Defined Networks GmbH
Berlin, Germany
Email: {tobi, marc.sune, hagen}@bisdn.de

***Abstract*—Network Functions Virtualization can enable each user (tenant) to define his desired set of network services, called *(network) service graph*. For instance, a User1 may want his traffic to traverse a firewall before reaching his terminal, while a User2 may be interested in a different type of firewall and in a network monitor as well. This paper presents a prototype of an SDN-enabled node that, given a new user connected to one of its physical ports, it is able to dynamically instantiate the user's network service graph and force all his traffic to traverse the proper set of network functions.**

***Keywords*-Network Service Graph; NFV Orchestration**

I. Introduction

Network Functions Virtualization (NFV) [1] proposes to transform the network functions (NFs) that today run on dedicated appliances (e.g., firewall, WAN accelerator), into a set of software images, which can be consolidated into high-volume standard servers thanks to the power of computing virtualization technologies and software switches. The unprecedented flexibility and agility provided by those technologies enable the creation of network service chains that can be finely customized, even on a per-user base, enabling each user to defined his own preferred set of network services, called *(network) service graphs*.

This paper focuses on the case in which users are given the possibility to define their own service graph [2], made up of a set of NFs chosen by the user himself and that operate solely on his traffic. The service graph is stored in a user profiles database and it is used when the network recognizes an user terminal that belong to the user himself, such as based on the MAC address of the terminal or on the physical port the user connects to. In this case, our system is able to reconfigure the network paths inside the node and instantiate the NFs (as specified in the service graph) on the user' traffic. This requires the network to translate the service graph, which includes only high-level information such as the chosen NFs and their service order, into a more precise representation called *Network Functions - Forwarding Graph* (NF-FG), which includes additional parameters (e.g. physical/virtual ports) required by the system to operate.

This paper presents a software architecture to dynamically instantiate NF-FGs starting from an high level description of the desired graphs and the occurrence of a particular event (e.g., a new user is connected to the node), leveraging traffic steering primitives provided by SDN technologies that are used to dynamically reconfigure the network paths inside the network node. The dynamic instantiation of NF-FGs is one of the topics of the EU-funded FP7 project UNIFY [3], which aims at providing full network and service virtualization to enable rich and flexible services.

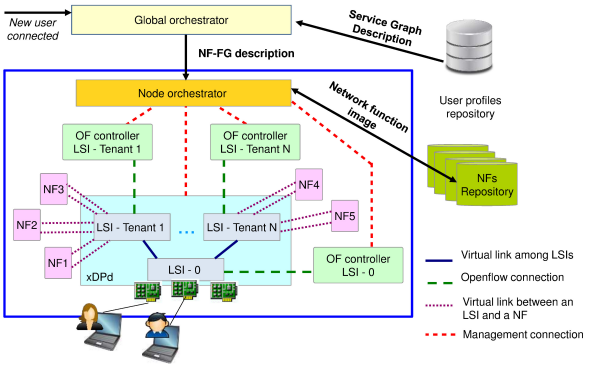

Figure 1. Overall architecture.

II. Architecture

The overall system architecture of our prototype is shown in Figure 1. The **global orchestrator** is the module in charge of authenticating a user connecting to the network node. Upon receiving the information that a new user terminal is connected to one of the physical ports of the node, the orchestrator retrieves the correct **Service Graph (SG)** stored in an external **user-profiles repository**, and outputs the corresponding **Network Functions - Forwarding Graph (NF-FG)** to the node orchestrator.

The NF-FG (Figure 2) is a JSON data that consists of a sequence of "flow-space/action" pairs, each one indicating which traffic has to be delivered to a specific NF (on a given virtual port), or the physical port through which this traffic has to leave the node itself. The flow space supports all the fields defined by Openflow 1.3 [4] (although new fields can be defined), while the action can refer either to a physical port of the node, or to a port of a NF. In this respect, we can consider the NF-FG as a generalization of the Openflow data model that specifies also the *functions* that have to process the traffic into the node, in addition to defining the (virtual) ports the traffic has to be sent to.

The **node orchestrator** receives the NF-FG through a REST API. This module takes care of instantiating the

```
"NF-FG" : {
  "id" : "user1_123",
  "flow-rules" :  [
  {
    "flow-space" : {
    "port"  : "eth0",
    },
    "action"  : "stateless_firewall:1"
  },
  {
    "flow-space" : {
      "port" : "stateful_firewall:2",
      "tcp_src" : "80"
    },
    "action" : "URLfilter"
  },
    .......
  ]
}
```

Figure 2. Excerpt of a NF-FG description.

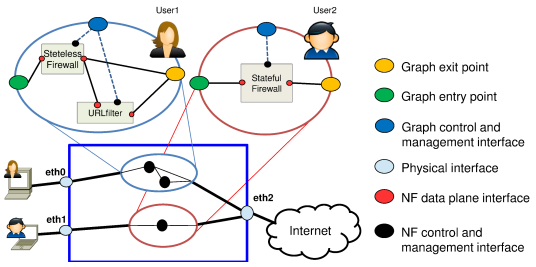

Figure 3. Example of NF-FG deployment.

NF-FG on the selected node, which requires to start the proper computing environments and to configure the network paths in the software switch. The implemented API also supports commands that delete/modify existing NF-FGs, while the extension to support cascading NF-FGs is currently in progress. When the node orchestrator receives a request for a new NF-FG, according to the capabilities of the node (e.g., number of CPU cores, specific hardware accelerators, etc.) it selects a NF implementation from a **NFs repository**, for each NF required by the NF-FG.

The traffic steering among the elements of the NF-FG is based on **xDPd** [5], a software switch that allows to dynamically create several software Openflow switches, called **Logical Switch Instances (LSIs)**, which can be connected to each other, to physical interfaces and to NFs. LSIs are required to provide the expected level of traffic isolation among users, allocating a distinct virtual switch to each user that is dedicated to the implementation of the service chaining of the user himself. Instead, the LSI-0 is in charge of classifying the user's traffic and delivering it to the proper LSI instance. LSIs access to the network ports using the DPDK framework [6]: the `igb` driver is used in case of physical interfaces, while the `kni` driver or `rte_rings` are used to exchange packets with NFs, according to the type of NFs themselves.

When the node orchestrator receives a new NF-FG description, it: *(i)* retrieves an implementation for each required NF (using the NFs repositories) and installs it, *(ii)* instantiates a `user-LSI` on xDPd, connecting it to the proper NFs and to the `LSI-0`, and then *(iii)* it creates a per-tenant **Openflow controller** that allows to insert the proper rules in the flow table(s) of the LSIs. In particular, rules defining a NF-FG are translated into two sequences of Openflow `flowmod` messages: one to be sent to the LSI-0, so that it knows which traffic must be provided to the `user-LSI` and how to treat packets coming from this LSI; the other to be sent to the `user-LSI`, to instruct it on how to steer packets between NFs.

The node supports two flavors of NFs: **DPDK processes**, and NFs deployed in **Docker containers** [7]. While the former type provides better performance (in fact, an LSI exchanges packets with DPDK NFs in a zero-copy fashion), Docker containers guarantee properties such as isolation among NFs, as well as they allow to limit the CPU and memory usage of NFs themselves. Finally, each NF is provided with an interface dedicated to the control and management of the NF itself, which can be used to download the proper configuration in each NF and/or to control its behavior.

III. Use case

Figure 3 provides a logical representation of a possible use case. *User1* configures a SG consisting of a stateless firewall (e.g., a Docker container running an `iptables` instance) and `URLfilter`, a DPDK process that identifies and drops HTTP requests belonging to a set of blacklisted URLs. Since the latter NF requires to operate on HTTP traffic, the flow-rules describing the graph will steer all non-HTTP traffic to `eth2`. *User2*, instead, configures a SG that just includes a stateful firewall, hence experiencing a network behavior that is different from the one defined for *User1*. Obviously this example can be straightforwardly enriched with more rules, NFs and users.

Acknowledgment

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union. Study sponsors had no role in writing this report. The views expressed do not necessarily represent the views of the authors' employers, the UNIFY project, or the Commission of the European Union.

References

[1] N. I. S. Group, "Network functions virtualization," Oct 2013. [Online]. Available: http://portal.etsi.org/NFV

[2] F. Risso and I. Cerrato, "Customizing data-plane processing in edge routers," in *Proceedings of the European Workshop on Software Defined Networking (EWSDN)*, 2012, pp. 114–120.

[3] A. Császár, W. John, M. Kind, C. Meirosu, G. Pongrácz, D. Staessens, A. Takács, and F.-J. Westphal, "Unifying cloud and carrier network: Eu fp7 project unify," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC '13)*.

[4] Openflow 1.3. [Online]. Available: https://www.opennetworking.org

[5] xdpd. [Online]. Available: http://www.xdpd.org

[6] Dpdk. [Online]. Available: http://dpdk.org

[7] Docker. [Online]. Available: http://www.docker.com